Process Patterns for Agile Methodologies

Samira Tasharofi[1] and Raman Ramsin[2]
1 University of Tehran, Department of Electrical and Computer Engineering
North Karegar, Tehran, Iran, stasharofi@ut.ac.ir
2 Sharif University of Technology, Department of Computer Engineering
Azadi Avenue, Tehran, Iran, ramsin@sharif.edu

Abstract. The need for constructing software development methods that have been tailored to fit specific situations and requirements has given rise to the generation of general method fragments, or *process patterns*. Process patterns can be seen in some third-generation integrated methodologies (such as OPEN) and in Method Engineering approaches where they are used as *process components*. They have also been presented as components in generic software development lifecycles where they represent classes of common practices in a specific domain or paradigm; object-oriented process patterns are well-known examples. Agile methodologies, however, are yet to be thoroughly explored in this regard. We provide a set of high-level process patterns for agile development which have been derived from a study of seven agile methodologies based on a proposed generic Agile Software Process (ASP). These process patterns can promote method engineering by providing classes of common process components which can be used for developing, tailoring, and analyzing agile methodologies.

1 Introduction

A *pattern* is a "general solution to a common problem or issue, one from which a specific solution may be derived" [1, 2]. *Process Patterns* are results of applying abstraction to recurring software development processes and process components; they are an effective mechanism for highlighting and establishing methods and approaches that have proven to be successful in practice [2].

Process patterns were first introduced by Coplien in 1994 [1], and were defined as "the patterns of activity within an organization (and hence within its project)". Coplien's patterns were relatively fine-grained techniques for exercising better organizational and management practices. Therefore, they did not constitute a comprehensive and coherent whole for defining a software development process.

Process patterns were later focused upon in the object-oriented paradigm. In his two books on object-oriented process patterns, Ambler defined an object-oriented process pattern as "a collection of general techniques, actions, and/or tasks

Please use the following format when citing this chapter:

Tasharofi, S., Ramsin, R., 2007, in IFIP International Federation for Information Processing, Volume 244, Situational Method Engineering: Fundamentals and Experiences, eds. Ralyté, J., Brinkkemper, S., Henderson-Sellers B., (Boston Springer), pp. 222-237.

(activities) for developing object-oriented software" [2, 3]. The proposed object-oriented patterns were categorized as belonging to three different types, commonly ordered by ascending level of abstraction and granularity as *tasks*, *stages*, and *phases*. A *task* process pattern depicts the detailed steps to execute a specific fine-grained task of a process. A *stage* process pattern defines the steps that need to be executed in order to perform a stage of the process and is usually made up of several task process patterns. Finally, a *phase* process pattern represents the interaction of two or more stage process patterns in order to execute the phase to which they belong. The process patterns introduced by Ambler constitute a proposed generic Object-Oriented Software Process (OOSP), which helps make sense of the relative position of the patterns in a general lifecycle, and their interrelationships. The approach relates to the one later put forward, in a more detailed and formal fashion, by Prakash [4]. Although these patterns have been intended to abstract common practices over a vast range of object-oriented methodologies, and are consequently rather general, their object-oriented-software-development nature makes them more tangible to software practitioners than Coplien's patterns.

Process patterns create means for developing methodologies through composition of appropriate pattern instances [5], a practice also commonly seen in *assembly-based* Situational Method Engineering [6, 7, 8]. One of the core elements in situational method engineering is a repository of reusable building blocks (also called method fragments or method chunks) from which method elements can be instantiated [9, 10]. Process patterns can provide a rich repository for the purpose of process assembly and/or tailoring. One of the main concerns with this repository is the provision of a good classification of building blocks so that it leads the method engineer to better selections. Classification of process patterns according to different domains of application (methodology types) can aid the method engineer in addressing this problem.

Process patterns have already been used to great effect in methodologies such as OPEN [11, 12] and are rapidly gaining popularity as process building blocks in method composition/configuration approaches such as the Rational Method Composer (RMC) [13]. Agile development, however, has enjoyed little attention in this regard: efforts have mostly been confined to Software Process Improvement (SPI) [14] and dual-methodology integration/customization [15]. A generic view on agile methodologies can only be seen in Ambler's proposed Agile System Development Life Cycle (ASDLC) [16], which is not only rather cursory in its treatment of the constituent process patterns, but also lacks ample coverage, as the abstraction and generalization it provides is mainly based on just two methodologies: XP [17, 18] and AUP [19].

In this work, we identify process patterns commonly encountered in agile methodologies. Because of common defining characteristics and basic underlying principles – as presented in [20] and set out in the Agile Manifesto [21] – Agile methodologies share many common constituents, which if extracted in terms of process patterns, can be used in constructing and/or tailoring other agile methodologies. In order to achieve this goal, we start from a generic model for agile software processes, which has resulted from inspecting seven prominent, widely-used agile methodologies. We then extract the recurring process patterns in a top-down fashion according to the three abstraction levels suggested by Ambler [2]. The

approach is similar to that applied in [2], yet the main contribution of our work is that the process patterns thus defined are agile-specific.

The organization of this paper is as follows: In Section 2, the proposed generic Agile Software Process (ASP) will be described. Section 3 introduces the process patterns derived from the ASP. Section 4 shows how different agile methodologies can be realized using the proposed patterns. Section 5 discusses the benefits obtained from the proposed agile process patterns, and finally, Section 6 contains the conclusions and suggestions for future work.

2 Agile Software Process (ASP)

The Agile Software Process (ASP), depicted in Fig. 1, is the proposed generic process model of agile methodologies. This model is obtained as a result of investigating seven agile methodologies: DSDM [22], Scrum [23], XP [17, 18], ASD [24], dX [25], Crystal Clear [26], and FDD [27].

ASP is composed of three serial phases which are in turn composed of internal iterative stages. According to these phases, an agile process begins with initiating the project; in the activities that follow, the software will be developed and deployed into the user environment through multiple iterations. In most agile methodologies, maintenance does not appear as a separate phase, but is rather performed through further iterations of the main development phases. Therefore, in ASP, maintenance is supported by a transition from the *Release* phase to the *Initiation* or the *Development Iterations* Phase. The other intention behind the transition from *Release* phase to *Development Iterations* phase is to accommodate frequent releases of software, which is followed as a principle in most agile methodologies.

The arrow at the bottom of the diagram indicates umbrella activities (expressed as task process patterns) which are critical to the success of a project and are applied to all stages of development. The phase and stage process patterns in ASP, as well as the tasks specified in the arrow, can in turn be detailed by delving into their constituent task process patterns.

ASP can be compared with Ambler's Object Oriented Software Process (OOSP) [2]. They are similar in several aspects: Some common stages can be found in their constituent phases, e.g., *Justify* and *Define Infrastructure*, and they are especially quite similar in the umbrella activities that they propose. But that is where the similarity ends. Since OOSP is proposed for all object-oriented methodologies regardless of their types, it is more general and consequently more abstract. This means that the patterns extracted from OOSP belong, more or less, to all object-oriented methodologies, whereas in ASP we have limited the extracted patterns to those found in agile methodologies. Therefore, ASP and OOSP are different in their structure and pattern content. The differences arise from the principles that define agility: For example, continuous verification and validation requires the existence of a review stage in the Development Iterations phase, the need for early and frequent releases of software necessitates the possibility of deploying working software increments into the user environment before deploying the complete system, and the change-based nature of agile methodologies has resulted in the absence of maintenance as a separate phase (as mentioned earlier). In the following sections,

the agile process patterns obtained from ASP are described in more detail. These patterns are classified, according to [2], as phase-, stage-, and task process patterns, and have been extracted from the generic ASP in a top-down fashion.

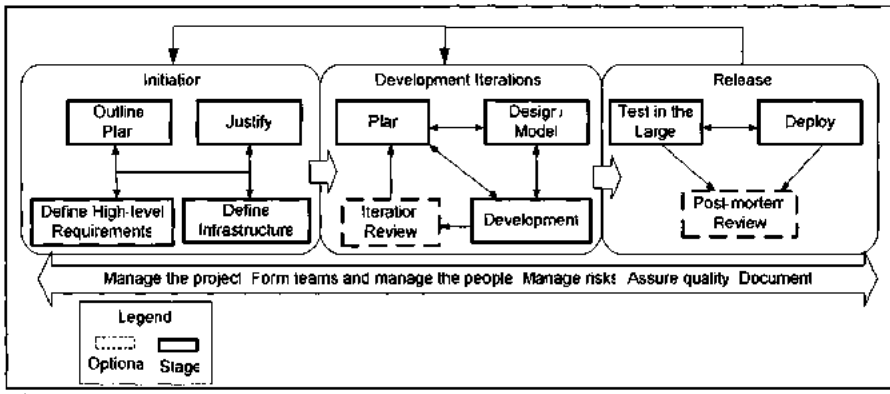

Fig. 1. The proposed Agile Software Process (ASP)

3 Agile Process Patterns

In this section, the agile process patterns extracted from the ASP will be described. For sake of simplicity and brevity, we use a more abstract notation than that used for describing object-oriented process patterns in [2]. For the same reason, we have avoided delving into the details of task process patterns and umbrella activities.

3.1 Phase Process Patterns

ASP consists of three phase process patterns: *Initiation*, *Development Iterations*, and *Release*. These are described below.

Initiation Phase
The goal in this phase is to initiate the project through preliminary analysis of the system. This phase consists of four iterative stages for providing an *outline plan*, *justifying* the project, and defining *high level requirements* and the *infrastructure* of the project.

Development Iterations Phase
In this phase, the working software is generated in multiple iterations. Each iteration is made up of planning, design, coding, testing and (optionally) review activities. These activities are covered by *Plan*, *Model/Design*, *Development*, and *Iteration Review* stage process patterns. As noted earlier, the transition from this phase to the *Release* phase and vice versa provides the possibility of deploying the newly generated software into the user environment after one or multiple iterations; the choice of the multiplicity depends on many factors, including the project type, and lies with the developer/manager.

Release Phase
Deployment activities of software engineering are performed in this phase. System-level testing (*Test in the large*) is done to verify and validate the system, and deliverable increments are deployed into the operational environment (*Deploy*). If it is revealed that the generated system satisfies its specification completely, or that the evolution of the system is impossible or unnecessary, the project will be terminated and may be reviewed by *Post-mortem Review*, in which the experiences obtained from the project are documented in order to be used in later projects. Otherwise, a return to *Initiation* or *Development Iterations phases* is required. As some agile methodologies (e.g., DSDM and Scrum) exclude post-mortem review, it has been specified as an optional stage.

3.2 Stage Process Patterns

Each phase in ASP is stated in terms of its constituent interrelated stages. Most of these stages can be performed iteratively. In this section, the stage process patterns of ASP are described in terms of their interrelated constituents – consisting of tasks and other nested stages – and the work products produced in and/or transferred among phases and stages.

Justify
In this stage (Fig. 2), the intention is to justify the project via a feasibility study and gain initial support and funding for the project.

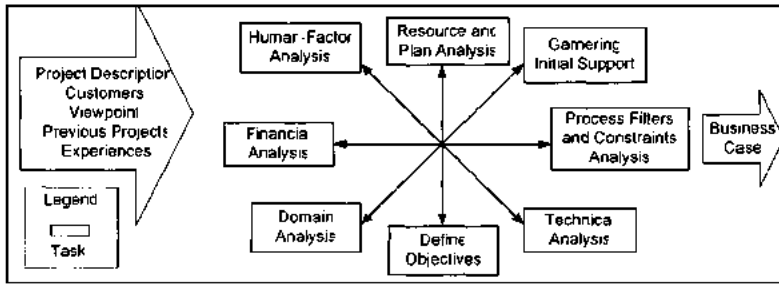

Fig. 2. Components of the *Justify* stage process pattern

As can be seen from Fig. 2, the input work products to this stage are *Project Description*, *Customers' Viewpoint* about the project, and the documented *Previous Projects Experiences*. The result of this stage is the project *Business Case* which represents the business value of the project. In this stage, feasibility study is performed through risk analysis which involves *Resource and Plan Analysis*, *Human-Factor Analysis*, *Financial Analysis*, and *Technical Analysis*. These tasks are coupled with the application of project constraints and process suitability filters in *Process Filters and Constraints Analysis*, *Defining Objectives*, and a domain walkthrough in the *Domain Analysis* tasks. At the end of this stage, customer approval and initial support for the start of the project will be obtained in the *Garnering Initial Support* task.

Define High-level Requirements

The requirements form the basis for other steps of the project. At the start of the project, initial high-level requirements are defined which will later be detailed and refined (Fig. 3). The required work products for this stage are *Project Description*, *Customers' Viewpoints* about the project, the *Business Case* defined in the Justify stage, and other related *Projects Experiences*; these documented experiences are often provided by the post-mortem reviews at the end of projects.

The requirements are identified and defined in *Problem Domain and Solution Domain Analysis*, and require active customer collaboration (*Get Customer Approval*). Examination of the problem and solution domains can be performed more precisely with the aid of modeling which is specified as the *Design/Model* stage in Fig. 3. A description of this stage will be given in the next section. Because of the model-phobic nature of many agile methodologies, this stage has been specified as optional. The products of this stage are a document of discriminated requirements (*Requirements Document*) and the generated models (*Models*).

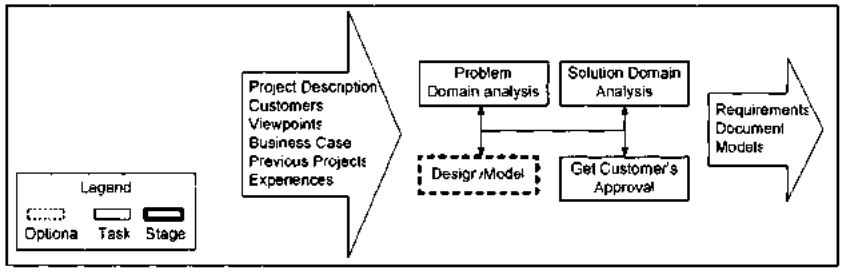

Fig. 3. Components of the *Define High-level Requirements* stage process pattern

Design/Model

Design and modeling may be used for defining and/or refining the requirements, the architecture, the design of the system, and the plans. Prototyping can also be considered as a task belonging to this stage. The iterative tasks of this stage, as depicted in Fig. 4, are defining the goal of design/modeling, designing and defining the alternatives, (optionally) using tools and prototyping to propose different alternatives, and reaching an agreement on the produced designs/models. The generated designs, models, and prototypes are packaged in the *Models* document.

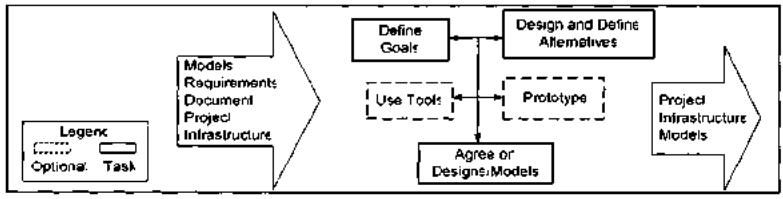

Fig. 4. Components of the *Design/Model* stage process pattern

Define Infrastructure

In this stage, project constraints, standards, and the system architecture are defined. As shown in Fig. 5, it uses the *Requirements Document*, *Business Case*, *Project Description* and *Previous Projects Experiences* to provide the *Project Infrastructure*.

This stage is performed through iterative tasks for defining rules and constraints, designing the architecture, specifying the development and operational platforms, defining goals and objectives, and (optionally) defining methodology conventions. The task *Define Methodology Conventions* is not found in all agile methodologies, yet it is considered an essential activity in some agile methodologies, such as Crystal. It has therefore been specified as optional. To define the system architecture, modeling, designing or prototyping may be needed. Therefore, the possibility of moving from *Define Architecture* to the *Design/Model* stage and vice versa has been accommodated. As a consequence of applying this stage, the requirements document may be changed or refined.

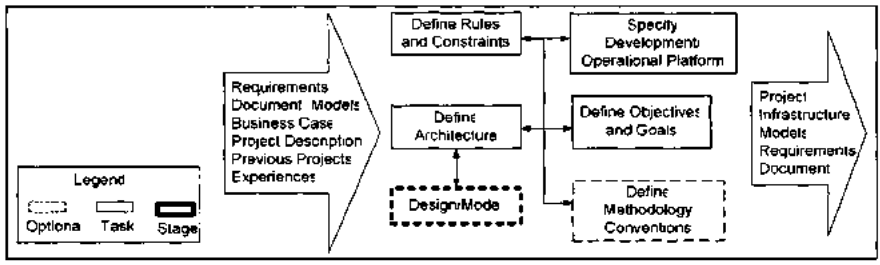

Fig. 5. Components of the *Define Infrastructure* stage process pattern

Outline Plan

In this stage the preliminary plan and schedule of the project are defined. As results of this stage, the initial project management document (*Management Document*) and the project plan and schedule (*Plan*) are produced. As deduced from Fig. 6, the required tasks include estimating the time, resources, and the effort needed for project completion, and preparing the management document according to these estimates. The management document contains all the information needed for project management (e.g., project schedule, plan, people communication paths, etc.). It may be needed to perform these tasks in multiple iterations. The requirements, project infrastructure, models, and previous projects experiences help refine the estimates.

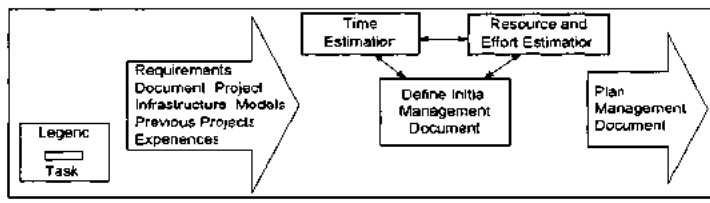

Fig. 6. Components of the *Outline Plan* stage process pattern

Requirements Analysis

The detailed analysis of requirements is carried out in this stage (Fig. 7). Existing requirements are refined and some new ones may also be added. Additionally, the requirements are prioritized according to different criteria depending on the project at hand, e.g., interdependencies, business value, or risks associated with the requirements. Designing and modeling can be used to gain a better understanding of the requirements. The requirements document is refined and completed in this stage.

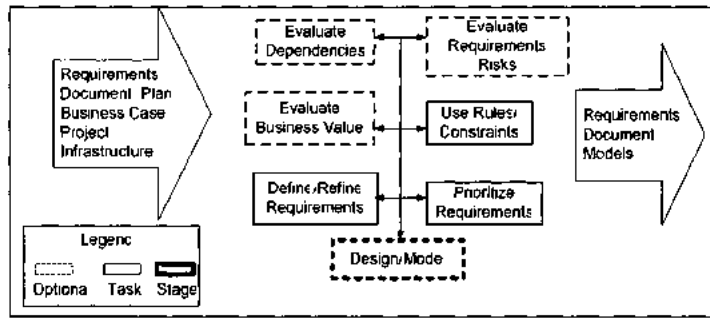

Fig. 7. Components of the *Requirements Analysis* stage process pattern

Plan

Because of frequent reviews in the development iterations of agile methodologies, the plan is likely to be refined or otherwise modified during the iterations. Therefore, at the start of each iteration in the *Development Iterations* phase, the project plan and schedule are reviewed and revised. This stage is shown in Fig. 8. The *Requirements Analysis* stage refines or otherwise changes the requirements document. Time boxes and artifacts of the next iteration(s) are then specified. The documented lessons learnt from previous iterations (*Iteration Review Document*), if existing, form an important artifact, based on which planning and scheduling decisions are made in this stage. The stage also involves the definition of tasks and their assignment to project team members. Some agile methodologies, e.g. DSDM, exclude defining and assigning tasks in each iteration; the two tasks are therefore specified as optional.

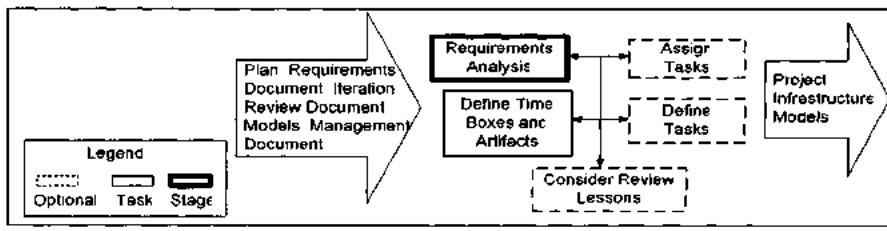

Fig. 8. Components of the *Plan* stage process pattern

Test in the Small

During the stages in which the system is evolved, the generated increments must be tested. These tests are not system-level, and specifically consist of unit testing, black-box testing, regression testing, and integration testing. Testing may be performed

with the aid of tools, as seen in XP. The constituent tasks and artifacts of this stage are demonstrated in Fig. 9. The requirements document is the basic artifact for this stage. If any test collections and documented results exist, they too will be used for regression testing or in repeating the failed tests. At the start of this stage, the goal of testing and the targets must be defined through planning the test. Test cases are then generated or may be selected from test collections according to the test plan. The results of running test cases are documented in the *Test Document* artifact. Because of active user involvement in agile methodologies, users may also test the product and give feedback to producers. While validation is a must in all projects, in some agile methodologies (e.g. Crystal Clear) this is done after multiple iterations, and not during each iteration. This is why the *User Test* task is specified as optional in this stage.

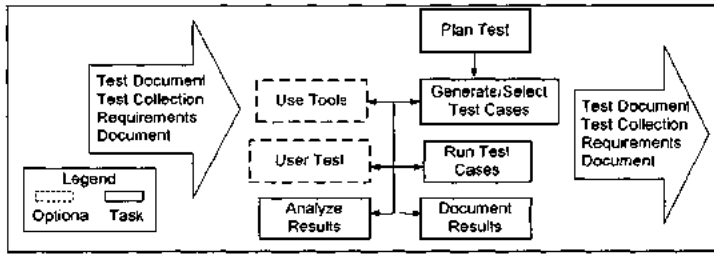

Fig. 9. Components of the *Test in the Small* stage process pattern

Test in the Large

This stage (Fig. 10) is where system-level testing is performed. The defects found in testing may be resolved by the *Fix Bugs* task in this stage, or deferred to the *Development Iterations* phase. The constituent tasks in this stage are similar to the *Test in the Small* stage with some differences: 1) the *User Test* task is not optional, 2) the defects found may be resolved in this stage, 3) Planning and generating test cases is based on system-level tests strategies, and 4) because of the need for bug fixing, the constituent tasks may be performed iteratively.

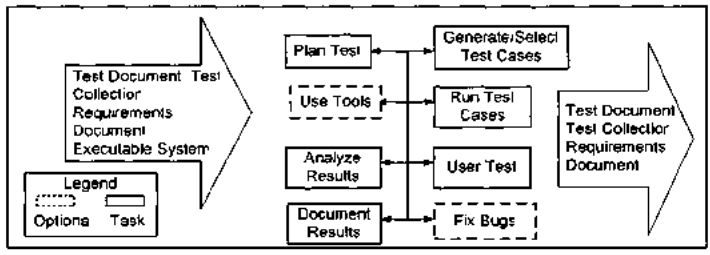

Fig. 10. Components of the *Test in the Large* stage process pattern

Review

Reviews play an important role in agile methodologies. Different types of reviews extracted from agile methodologies are: *Product Review*, *Process/Plan Review*, and *Project Review* (*Post-mortem Review*).

Product Review

The product is reviewed via analyzing the test results, validating the product through delivery to customers, comparing the results with defined goals, and documenting the conclusions in the *Product Review Document* (Fig. 11). As a consequence, the requirements and project infrastructure may be changed.

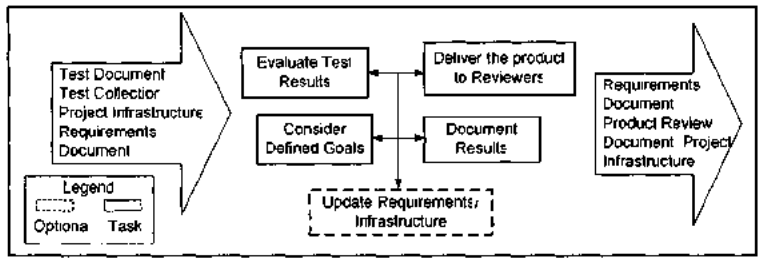

Fig. 11. Components of the *Product Review* stage process pattern

Process/Plan Review

Process/Plan Review, as shown in Fig. 12, aims at adapting the applied process/plan with the current state of the project. The plan of the project, management document, project infrastructure, and product review document help assess the process/plan. Therefore, the project plan and schedule must be compared with the current state of the project and the project velocity, the encountered problems must be analyzed, and the tuning points of the process/plan must be specified. The results are recorded in the *Process/Plan Review Document*.

Post-mortem Review (Project Review)

At the end of the project, the project will be investigated and the lessons learned are documented in the *Post-mortem Review Document*. This stage, as illustrated in Fig. 13, uses the product- and process/plan review documents, management document, project plan and infrastructure to make a tour of the system, compare the initial estimates with the current state of the project, using users' opinion on the system, and analyze the problems and solutions. This stage provides a good protection against the *Reinvent the Wheel* process antipattern [28].

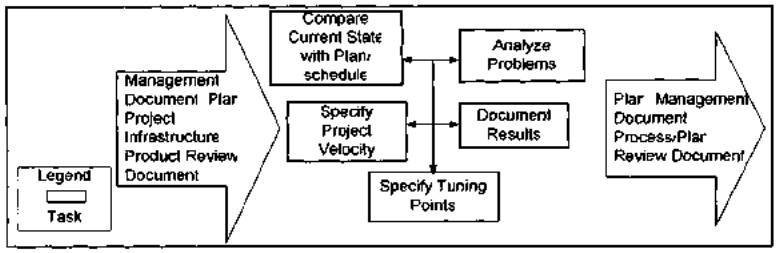

Fig. 12. Components of the *Process/Plan Review* stage process pattern

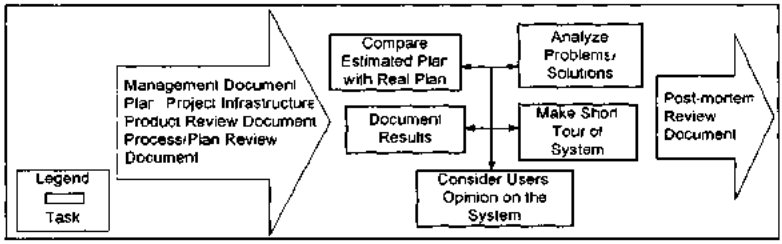

Fig. 13. Components of the *Post-mortem Review* stage process pattern

Implement

Implementing the requirements and resolving the defects are performed in this stage (Fig. 14).

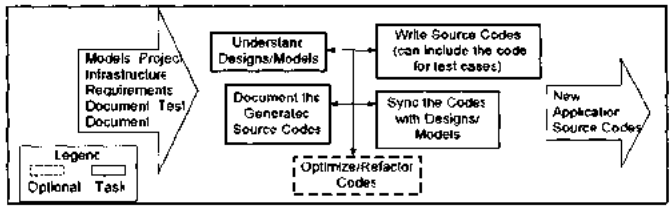

Fig. 14. Components of the *Implement* stage process pattern

The test document is used for fixing the bugs diagnosed during test activities. The generated code (which may include the test code, as commonly seen in Test-Driven Development) must conform to the requirements/defects and models/designs. Code inspection with the aim of refactoring and code optimization is a practice used in most agile methodologies after generating the source code (e.g., XP and ASD). The outcome of this stage is a new version of the product.

Integrate

Integration of newly generated increment(s) with the current system is handled in this stage (Fig. 15). Inputs to this stage are the new increment, the current integrated system, and the project infrastructure which contains the standards and constrains governing integration. The environment must first be prepared for the new increment; the new application is then integrated with the current system (this may

be done iteratively) and the new system is prepared for testing (e.g., integration test, regression test, etc.). The strategy governing the time and frequency of integration is dependent on the nature of the process, the plan, and the project itself.

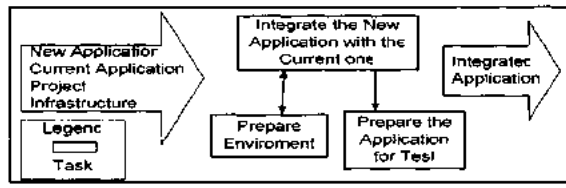

Fig. 15. Components of the *Integrate* stage process pattern

Deploy

This stage, as seen in Fig. 16, is made up of all the tasks related to the deployment of the system into the user environment. It consists of setting up the user environment, deploying the system, preparing user documents, and training the users. Tasks must be performed with attention to the constraints delineated in the project infrastructure.

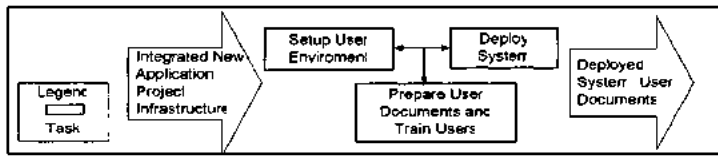

Fig. 16. Components of the *Deploy* stage process pattern

Iteration Review

This type of review is carried out after performing the iteration(s) in the *Development Iterations* phase. The aim is to adapt the plan and the process with the project and the development team in order to enhance product quality. Therefore, as shown in Fig. 17, it consists of *Process/Plan Review* and *Product Review* stages performed in an iterative manner.

Development

This stage (Fig. 18) is preformed via iterative application of the *Implement*, *Test in the Small*, and *Integrate* stages. The input and output work products are the union of the inputs and outputs of the constituent stages. The goal is to produce, integrate and test different parts of the system.

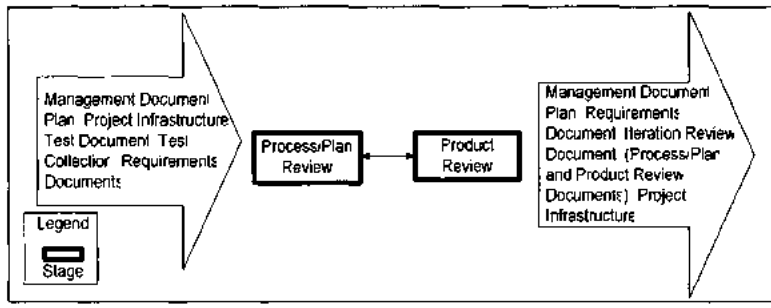

Fig. 17. Components of the *Iteration Review* stage process pattern

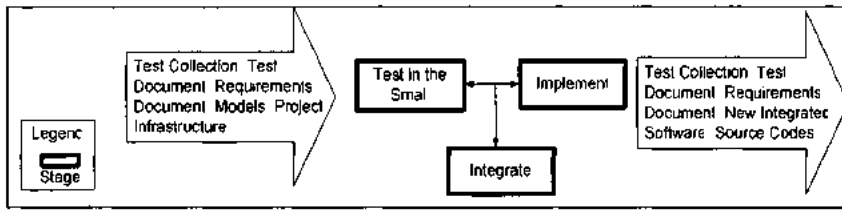

Fig. 18. Components of the *Development* stage process pattern

4 Realization of the Proposed Process Patterns in Agile Methods

Table 1 shows how different phases of the agile methodologies studied in our work can be realized by our suggested process patterns. We abstract away from umbrella activities, e.g., people and project management, which span all phases of the project and correspond to task process patterns. Therefore, they do not appear in this table. The realization table has been used to verify that the extracted process patterns indeed cover the methodologies used as the bases.

5 Applications of the Proposed Agile Process Patterns

The Agile process patterns proposed herein can facilitate situational method engineering (SME) when aimed at constructing agile methodologies to match given organizational settings or specific development projects [8, 29]. The process patterns can be used in the *assembly-based* approach of SME [29, 9] as classes of agile method chunks which can be used for composing agile methodologies. Furthermore, ASP and agile process patterns can be used in enacting the *paradigm-based approach* of SME [29] for instantiation and adaptation of process and product models.

The process patterns can also provide the basis for a rich component library for instantiating process components according to a predefined framework, typically depicted as a method engineering meta-model; much in the fashion of OPEN/OPF [11, 12].

Table 1. Realization of the proposed agile process patterns in agile methodologies

Methodology	Phases	Corresponding Stage Process Patterns
DSDM	Feasibility Study	Justify, Outline Plan
	Business Study	Define High-level Requirements, Plan, Define Infrastructure
	Functional Model	Plan, Requirements Analysis, Design/Model
	Design and Build	Plan, Development, Design/Model
	Implementation	Test in the large, Deploy, Product Review
	Post-Project	Further iteration of previous five main phases
Scrum	Pre-game: planning	Define High-level Requirements, Requirements Analysis, Outline Plan
	Pre-game: Architecture/ High-level Design	Define Infrastructure
	Development	Plan, Process/Plan Review, Design/Model ,Implement, Test in the Small, Iteration Review
	Post-game	Integrate, Test in the large, Deploy
XP	Exploration	Define High-level Requirements, Define Infrastructure
	Planning	Outline Plan, Plan
	Iterations to First Release	Plan, Design/Model, Development, Process/Plan Review
	Productionizing	Deploy, Test in the Large
	Maintenance	Repetition of three previous phases
	Death	Post-mortem Review
ASD	Project Initiation	Justify, Define High-level Requirements, Define Infrastructure
	Iterative Development Phases	Plan, Development, Iteration Review
	Final Q/A and Release	Test in the large, Deploy, Post-mortem Review
dX	Inception	Define High-level Requirements, Define Infrastructure, Outline Plan
	Elaboration	Plan, Design/Model, Development, Iteration Review
	Construction	Plan, Design/Model, Development
	Transition	Deploy, Test in the large
Crystal Clear	Chartering	Justify, Outline Plan, Define High-level Requirements, Define Infrastructure, Plan
	Delivery Cycle	Plan, Design/Model, Development, Iteration Review
	Wrap-up	Test in the large, Deploy, Post-mortem Review
FDD	Develop an Overall Model	Design/Model
	Build a Features List	Requirements Analysis
	Plan by Feature	Plan
	Design by Feature	Design/Model
	Build by Feature	Implement, Test in the Small

Finally, because of their abstract nature, the proposed process patterns lend themselves better to adaptation and tailoring, thereby enhancing configurability and dynamic flexibility; a feature which can be indispensable in agile methodologies, where the process itself needs to be adaptable based on the circumstances surrounding different project situations.

6 Conclusion

We have proposed a set of agile-specific process patterns that can be used for method engineering purposes. Pattern extraction was based on detailed inspection of seven prominent agile methodologies, and a generic Agile Software Process (ASP) was identified and used as the starting point for the extraction process. We have also demonstrated how each studied agile methodology can be realized using the proposed process patterns. Our suggested process patterns thus provide classes of reusable agile process building blocks that can be instantiated and used for composing and tailoring agile processes.

This work can be further extended to investigate the full details of the task process patterns, and especially address the umbrella activities covered in the generic ASP. Future work can then be directed towards developing a Computer Aided Method Engineering (CAME) environment [7, 30] that facilitates assembly-based engineering of agile methodologies using the agile process patterns introduced herein as reusable method fragments stored in a method base. ASP's role will be that of a generic method model providing a general template for agile methodologies, thus adding the support for paradigm-based SME. Another strand can focus on defining extension points for agile process patterns, further layering the patterns architecture into *core* process patterns and available *extensions*, thereby enhancing complexity management and promoting the production of lighter methodologies.

Acknowledgment. We wish to thank the Research Vice-Presidency of Sharif University of Technology for sponsoring this research.

References

1. J. O. Coplien, A Generative Development Process Pattern Language, in: Pattern Languages of Program Design (ACM Press/Addison-Wesley, 1995), pp. 187-196.
2. S. W. Ambler, Process Patterns: Building Large-Scale Systems Using Object Technology (Cambridge University Press, 1998).
3. S. W. Ambler, More Process Patterns: Delivering Large-Scale Systems Using Object Technology (Cambridge University Press, 1999).
4. N. Prakash, On generic method models, Requirements Engineering 11(4), 221-237 (September 2006).
5. K. Bergner, A. Rausch, M. Sihling, and A. Vilbig, A Componentware Development Methodology based on Process Patterns, in: Proceedings of PLoP-98 (1998).
6. K. Kumar and R. J. Welke, Method engineering: a proposal for situation-specific methodology construction, in: Systems Analysis and Design: A Research Agenda, (Wiley, 1992), pp. 257-268.
7. A. F. Harmsen, Situational Method Engineering (Moret Ernst & Young, 1997).
8. I. Mirbel and J. Ralyté, Situational method engineering: combining assembly-based and roadmap-driven approaches. Requirements Engineering 11(1), 58–78 (March 2006).
9. S. Brinkkemper, M. Saeki and F. Harmsen, Assembly techniques for method engineering. in: Proceedings of CAiSE'98 (1998), pp. 381–400.
10. S. Brinkkemper, Method engineering: Engineering of information systems development methods and tools. Information and Software Technology 38(4), 275–280 (Apr.1996).
11. D. Firesmith and B. Henderson-Sellers, The OPEN Process Framework: An Introduction (Addison-Wesley, 2001).
12. B. Henderson-Sellers, Method Engineering for OO Systems Development, Communications of the ACM 46(10), 73-78 (October 2003).

13. P. Kroll, Introducing IBM Rational Method Composer, published on the web at: http://www-128.ibm.com/developerworks/rational/library/nov05/kroll (2005).
14. B. Henderson-Sellers and M. K. Serour, Creating a dual-agility method: The value of Method Engineering, Journal of Database Management 16(4), 1-23 (Oct./Dec. 2005).
15. B. Fitzgerald, G. Hartnett and K. Conboy, Customizing agile methods to software practices at Intel Shannon, European Journal of Information Systems, 15(2), 200-213 (April 2006).
16. S. W. Ambler, The agile system development lifecycle, published on the web at: http://www.ambysoft.com/essays/agileLifecycle.html (2006).
17. D. Wells, Extreme programming: A gentle introduction, published on the web at: http://www.extremeprogramming.org (2006).
18. K. Beck and C. Andres, Extreme Programming Explained: Embrace Change, 2nd Ed (Addison-Wesley, 2004).
19. S. W. Ambler, The agile unified process, published on the web at: http://www.ambysoft.com/unifiedprocess/agileUP.html (2005).
20. D. Turk, R. France and B. Rumpe, Limitations of agile software processes, in: Proceedings of XP (2002), Alghero, Italy.
21. K. Beck, et al, Manifesto for agile software development, published on the Web at: http://agilemanifesto.org (2001).
22. DSDM Consortium, J. Stapleton, DSDM: Business Focused Development, 2nd Ed. (Addison-Wesley, 2003).
23. K. Schwaber and M. Beedle, Agile Software Development with Scrum (Prentice-Hall, 2001).
24. J. Highsmith, Adaptive Software Development: A Collaborative Approach to Managing Complex Systems (Dorset House, 2000).
25. G. Booch, R.C. Martin and J. Newkirk, Object Oriented Analysis and Design with Applications, 2nd ed. (1998), (Unpublished).
26. A. Cockburn, Crystal Clear: A Human-Powered Methodology for Small Teams (Addison-Wesley, 2004).
27. S. R. Palmer and J. M. Felsing, A Practical Guide to Feature-Driven Development (Prentice-Hall, 2002).
28. W. J. Brown, R. C. Malveau, H. McCormick, T. Mowbray, Antipatterns: Refactoring Software, Architectures, and Projects in Crisis (Wiley, 1998).
29. J. Ralyté, R. Deneckére and C. Rolland, Towards a generic model for situational method engineering, in: Proceedings of CAiSE2003 (2003), pp. 95-110.
30. S. Kelly, K. Lyytinen, M. Rossi, MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment, in: Proceedings of CAiSE'96 (1996), pp. 1-21.